CM-004

DID（分散型識別子）システムを用いた パーソナルデータアプリケーションの実装手法

江口　静 [1]　　道方　孝志 [2]　　越塚　登 [2]

[1] 東京大学大学院学際情報学府
[2] 東京大学大学院情報学環・学際情報学府

1　はじめに

現在の ID 管理は, 非常に複雑であるが, 大きく分けると集中型モデルとフェデレーションモデルに分類することができる.

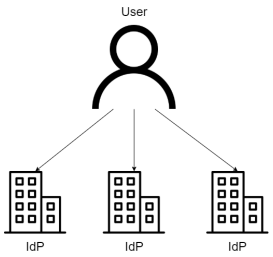

図 1: 集中型モデル

図 1 の集中型モデルでは, サービスを提供する Relying Party と ID を提供する Identity Provider (IdP) が同一となり, 一つの主体がユーザーのアイデンティティを管理しサービスを提供する. 例えば, ユーザーは email とパスワードでの認証をサービスごとに行う.

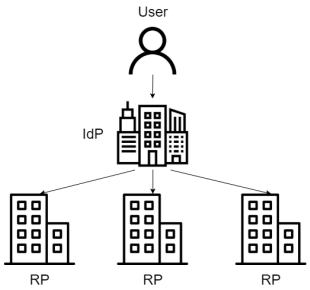

図 2: フェデレーションモデル

一方、フェデレーションモデル（図 2）では, Relying Party と Identity Provider は, 別の主体であり, ユーザーは, Relying Party のサービスにアクセスするときは, Identity Provider が提供するアイデンティティを利用する. ユーザーは Relying Party ごとにログインする必要はなく, 特定の Identy Provider が提供するアイデンティティ情報をもとに複数のサービス主体にアクセスすることができる.

フェデレーションモデルでは, 個人情報漏洩のリスクを Identity Provider に依存しているという問題がある. 実際に, Facebook に登録されていたユーザーの個人情報が, ハッキングによって外部に漏れたという事例がある[1].

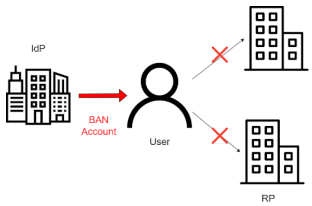

図 3: アカウント停止

また、 ユーザーのアイデンティティは Identity Provider の意向に左右されるという問題もある。例えば、 図 3 のように SNS のアカウントを停止させられると, そのアカウントを使ってログインしていた他のサービスも同時に使えなくなってしまうことが挙げられる.

こうした課題を解決するために, 近年新しい ID の仕組みとして, DID（Decentralized Identifier）が提案されている。DID とは個人が分散的に ID（Identifier）を管理するために作られた規格である. DID は現在の ID 管理の問題を解決する手段として有力視されている一方で, これまでアプリケーションレベルで実装されているものは少ない. 本研究の目的は, DID を利用するアプリケーションの要件や応用可能性を明らかにし、それらの要件や可能性を満たす DID の実装方法及び、DID を用いたアプリケーションや応用システム実装方法を提案することである。

[1] https://www.bleepingcomputer.com/news/security/533-million-facebook-users-phone-numbers-leaked-on-hacker-forum/

そこで、本論文では, 初めに, 一般的に利用されている ID 管理と比較しつつ, DID の概念や背景となる思想, その仕組みについてまとめる．次に, DID の活用モデルの一つである Verifiable Credentials Data Model を活用事例とともに取り扱うことでデジタルサービスにおける DID 利用の流れを明らかにする．最後に, より柔軟で汎用性が高いブロックチェーンフレームワークである Cosmos-SDK 及び Tendermint とそのスマートコントラクトモジュールである CosmWasm を用いて DID の実装[2]方法を提案する。更に、当該実装を通して得た知見をもとに, DID を利用するアプリケーションの要件をまとめ, DID の応用可能性を考察する．

2 DID の概要

2.1 ブロックチェーン

ブロックチェーンとは, 狭義には, Bitcoin[1] の基盤となる技術をさし, Proof of Work (PoW) と呼ばれるアルゴリズムを採用し CPU の演算能力を使用することで, ビザンチンノードを含む不特定多数のノード間で合意形成を行う仕組みのことである．また, 電子署名とハッシュポインタを使用し改竄検出が容易なデータ構造を持ち, 且つ, 当該データをネットワーク上に分散する多数のノードに保持させることで, 高可用性及びデータ同一性等を実現する技術を広義のブロックチェーンと呼ぶ [2].

2.2 DID の背景・目的

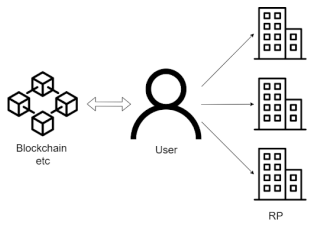

図 4: SSI の概念図

関連研究の前提として, DID が誕生した背景にある思想や DID の特徴について説明する．現在の ID 管理の問題点に対して, 個人情報は第三者の管理主体を介することなく, 個人が主権的にコントロールすべきであるという思想が生まれた [3]．この思想のことを SSI (Self Sovereign Identity) と呼ぶ．ユーザーのデジタルアイデンティティが特定の Identity Provider に依存しないよう, その依存度を下げることを目的としている．図 4 のように, ユーザーは, 自分のアイデンティティを自分で管理し選択的に Relying Party に提供することができる．ユーザーのアイデンティティの根幹である DID は, 特定の管理者がいないブロックチェーン等の分散ネットワークで管理されている．

2.3 DID とは

DID とは, 個人が分散的に ID を管理するために作られた規格のことである．World Wide Web Consortium (W3C)[3] や Decentralized Identity Foundation (DIF)[4] が中心となって規格作成を行っている [4]．DID の概念が登場したのは, 個人情報は個人が主権的に管理すべきであるというさきほど説明した SSI の思想が背景にある．この DID を活用したデータモデルとして, 後述する Verifiable Credentials などがある．

2.4 DID の特徴

DID は, 既存の Identity Provider のような一元化された登録機関を必要としない代わりに, ブロックチェーンまたは, 他の形式の分散型ネットワークを利用している．また, DID は, 暗号的に生成・登録される, グローバルに一意な永続的な識別子として機能する．

2.5 Verifiable Credentials Data Model

関連研究として W3C が定義している DID の活用モデルについて説明する．今回紹介するのは, Verifiable Credentials Data Model と呼ばれるもので, 現在, DID を利用するアプリケーションの一番有力なモデルとなっている [5]．Verifiable Credentials とは, 検証可能な個人情報といった意味で, Credentials の例としては, 運転できるという証明及び許可を表す運転免許証や, 教育レベルの証明としての卒業証書, 政府による国家間の移動を保障するパスポートなどが挙げられる．このモデルでは, DID を利用して, このような個人情報を暗号学的に安全で, プライバシーを保護しつつ, 検証可能な状態で, web 上でどのように表現するのかを表している。

[2] https://github.com/EG-easy/did-contract

[3] https://www.w3.org/

[4] https://identity.foundation/

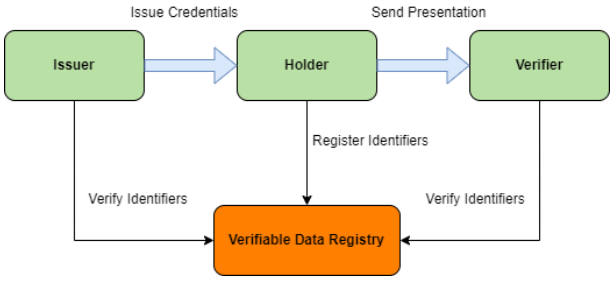

図 5: Verifiable Credentials Data Model の概要

Verifiable Credentials Data Model の一例を図 5 に示した。まず, アイデンティティを管理する Holder と Credential を発行する Issuer, Issuer が発行した Credential を Holder が保有していることを検証する Verifier の 3 者が存在する．また, Verifiable Data Registry は, ブロックチェーンのような検証可能で分散的に管理されたデータベースを指す．Holder は, Issuer から Credential を発行されると, それを第三者の Verifier に渡して自分が Credential の正当な保有者であることを明らかにすることができる．

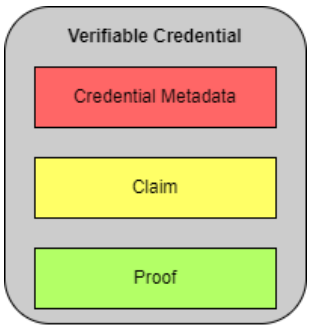

図 6: Verifiable Credential のデータ構造

図 6 は, Verifiable Credential の概要を表現している．Credential Metadata には, Credential の発行を行う Issuer や Credential の対象, その他 Credential に関する外的情報を格納している．Claim には, Holder に関する情報群が書き込まれる．Proof では, Credential の発行者が Credential Metadata に対して署名を行い, 検証可能な状態を付与している．

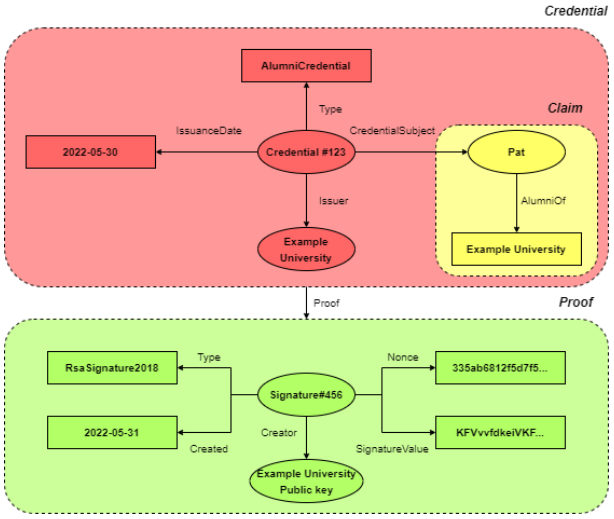

図 7: Verifiable Credential のデータフロー

図 7 は, 大学の卒業証明に Verifiable Credential Data Model を適応した場合のデータフローを示している．Holder は, Pat, Issuer は Example University になる．Example University は, Pat が卒業生であることを記した Credential を発行する．このとき, その Credential 情報に対して, Example University が秘密鍵で署名を行うことで, Credential が Example University の名において発行されたことがわかる．

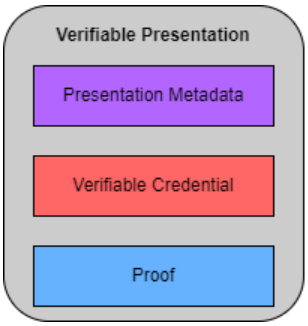

図 8: Presentation フローのデータ構造

さらに, Credential の Holder は, 自分の Verifiable Credential に Presentation Metadata を付与し、自分の秘密鍵で署名を行い第三者に渡すことも可能である．図 8 は, Credential を第三者にわたす際のデータ構造を示している。

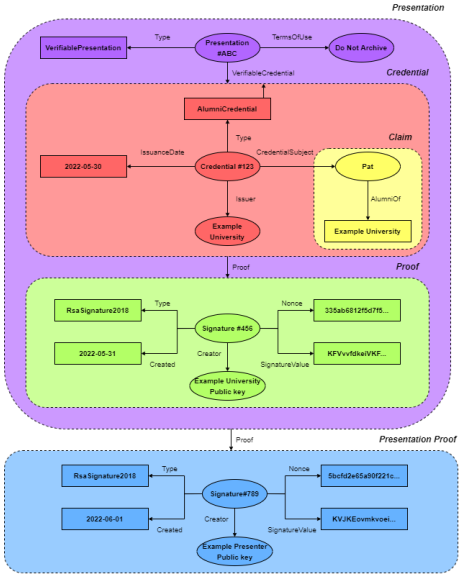

図 9: Presentation のデータフロー

図 9 に, さきほどの例を活用して Credential 情報を第三者に渡す流れ示す. Holder である Pat は, Example University が署名した Credential 情報に, 自分の秘密鍵で署名して第三者に提供する. 第三者は, Pat の公開鍵および Example University の公開鍵を使って署名済みのメッセージを復号すると, その Credential が Example University から発行され, Pat が保有しているということを検証することができる. 一例として, この第三者は, この仕組みを利用して, 卒業生のみが参加できるアルムナイサービス等を運営することができる.

2.6 DID の実装例

DID の仕様は, W3C が公開している[5]. その仕様に沿って, 実際にこれまで実装されたものの多くは, Ethereum や Hyperledger といったプラットフォームを利用している. 例えば, Sovrin[6]は, Hyperledger 上のプロジェクトで, Uport[7]は, Ethereum 上のプロジェクトである.

3 DID の課題

DID は現在の ID 管理の問題を解決する手段として有力視されている一方で, 現状では, W3C による仕様策定段階であり, 次のような課題が残されている.

- 限られたプラットフォーム上での実装しかない
- DID を使ってアプリケーションレベルで実装されているものが少ない

[5]https://www.w3.org/TR/did-spec-registries/
[6]https://github.com/sovrin-foundation/sovrin
[7]https://github.com/uport-project/ethr-did

4 DID の実装

本研究では, ブロックチェーン側でより柔軟な設計や実装が可能であり, アプリケーションごとにスケーラビリティを確保できる Cosmos-SDK[A] と Tendermint[B] の組み合わせで土台となるブロックチェーン部分を構築し, CosmWasm[C] のスマートコントラクトモジュール上で DID を実装した. この CosmWasm を使った DID 実装はまだ行われていないため, 本研究の新規性の一部となる. また, Cosmos-SDK や Tendermint を利用した DID の実装で確立したものはなく, 議論が行われているに過ぎないというのが現状である[8].

従って、本研究において従来とは異なるブロックチェーンプラットフォーム上で DID の実装を行うことで、DID の汎用性を高めるとともに, 実際にアプリケーションを構築することで DID を利用するアプリケーションの要件や応用可能性を明らかにすることができる。

5 DID の実装方法

CosmWasm を用いたスマートコントラクト開発は, EIP1056[9]の実装を参考にして行った. 具体的には, スマートコントラクト上に Map 上のデータ構造で Identity とその Owner を保存し, Event を発火させることで, 各メタデータの追加・変更・削除の履歴を追えるような構成した. ここで, Identity と Owner はブロックチェーンで使用されるアドレスと同一の形式になる. 具体的には, 表 5 の 4 つのメソッドを実装することで, DID および DID Document に対応した.

表 1: Method 一覧

Method Name	Type
identity_owner	Query
change_owner	Execute
set_attribute	Execute
revoke_attribute	Execute

また, 表 5 のメソッドを実行するときに必要なメッセージ（引数）は次の通りである.

```
{
  "identity_owner": {
    "identity": "wasm1fp7rrd...2f0yyq"
  }
}
```

Listing 1: Identity Owner Message

[8]https://github.com/cosmos/cosmos-sdk/discussions/8431
[9]https://github.com/ethereum/EIPs/issues/1056

identity_owner は, Query 型のメソッドで, Identity の Owner を返す. デフォルトでは, Identity の Owner は, Identity 自身なので, 自分の Identity アドレスを返す. 一方, Identity の Owner が変更されていた場合, 変更後の Owner のアドレスを返す.

```
{
  "change_owner": {
    "identity": "wasm1fp7rrd...2f0yyq",
    "new_owner": "wasm1rhhjhf...y2kn7y"
  }
}
```

Listing 2: Change Owner Message

change_owner は, Execute 型のメソッドで, Identity の Owner を変更する. デフォルトの状態では, Identity の Owner は, Identity 自身で, Identity のアドレスの秘密鍵を知っている主体のみが Identity の Owner を変更することができる. この関数を実行できるのは, Identity の Owner のみである.

```
{
  "set_attribute": {
    "identity": "wasm1fp7rrd...2f0yyq",
    "name": "example property",
    "value": "abc",
    "validity": 86400 // 1 day
  }
}
```

Listing 3: Set Attribute Message

set_attribute は, Execute 型のメソッドで, Identity に紐づくメタデータを記録する. メタデータの形式は, name-value の形でどのような property も書き込むことができる. また, このパタメータの有効期限を定めることができる. この関数を実行できるのは, Identity の Owner のみである.

```
{
  "revoke_attribute": {
    "identity": "wasm1fp7rrd...2f0yyq",
    "name": "example property",
    "value": "abc"
  }
}
```

Listing 4: Revoke Attribute Message

revoke_attribute は, Execute 型のメソッドで, Identity に紐づくメタデータを無効にする. メタデータの形式は, name-¿value の形書き込まれた property の有効期限を 0 にすることで, 無効化する. この関数を実行できるのは, Identity の Owner のみである.

5.1 ERC1056 ベースの DID の実装のメリット

ブロックチェーン上にデータを書き込む際には, 手数料が発生するため, Identity を新規作成する際に毎回ブロックチェーン上に書き込む方式の場合, そのコストは課題となる. 今回の実装では, ブロックチェーン上のアドレスを Identity として利用することで, アドレス=Identity となり, オフチェーン上でも利用できる Identity を手数料なく作ることができる. ユーザーは, DID の所有者の変更履歴や DID Document を追加破棄する際にブロックチェーンにデータを書き込むときに初めて手数料を支払う. また, この Identity は, 通常のウェブサービス内でも使うことができる仕組みになっている.

5.2 パーソナルデータ・アプリケーションの設計要件

ブロックチェーンに書き込むデータ設計

どのようなデータをどのような形式でオンチェーン上に記録するのかというのを考える必要がある. ブロックチェーンのデータ構造は, 本質的に Key-Value の構造をとるため, 従来の RDB (Relational Database) のようなデータ構造を持たせることは難しい. 5 章でみてきたように, 今回の実装では, DID の Owner を Owners[Identity] の Key-Value 構造で保存している.

オフチェーン設計

ブロックチェーンに書き込まれたデータをオンチェーンデータに対して, ブロックチェーン以外のところで保持しているデータのことをオフチェーンデータと呼ぶ. 実サービスで運用していくにあたり, RDB の, 関連する ID に関するデータを一括で取得できる機能は UX を向上させるためには必要不可欠である. また, Index を使って大量のデータから効率よく必要なデータを抽出する機能も使いたい. このようなオフチェーンデータの設計を行うことが通常のウェブサービス同様に重要になる.

API **設計**

通常, ブロックチェーンの現在の状態を知りたい場合は, ブロックチェーンノードに対して直接 Query を発行してデータを取得する. 一方で, 過去のデータを参照したい場合は, ブロックチェーンから必要な情報を同期したオフチェーンのデータベースを参照するほうが便利な場合が多い. API の設計もブロックチェーンを直接参照する場合と, オフチェーンデータベースを参照する場合の 2 通りを念頭に置く必要がある.

オフチェーンデータとオンチェーンデータの検証可能な設計

オフチェーンデータは, オフチェーン管理者によって恣意的に変更される可能性があるため, 第三者がオフチェーンデータの内容をオンチェーンでも確認できる仕組みを作っておく必要がある. 例えば, 今回の場合, DID の attribute が変更された場合には, CHANGED のプロパティにて変更されたときのブロック高が記録されているため, オフチェーンデータベースでもこのブ

ロック高を記録しておく必要がある．ユーザーは，オフチェーンデータのブロック高から実際に発生したトランザクションをオンチェーンで探し出し検証作業を行うことができる．

5.3 アプリケーション実装時の留意点

ブロックチェーンには，event と呼ばれる機能があり，トランザクションが発行されるときに，プロパティを指定してそのブロック高時点での情報を記録することができる．今回の DID の実装では，DID に紐づく attribute が更新されたとき，および Identity の Owner が変更されたときに下記のように event が発行される．

```
[
    {
        "key": "identity",
        "value": "wasm1fp7rrd...2f0yyq"
    },
    {
        "key": "name",
        "value": "example property"
    },
    {
        "key": "value",
        "value": "abc"
    },
    {
        "key": "validTo",
        "value": "1655948862.138366570"
    },
    {
        "key": "previousChange",
        "value": "0"
    },
    {
        "key": "from",
        "value": "wasm1fp7rrd...2f0yyq"
    }
]
```

Listing 5: DIDAttributeChanged Event

```
[
    {
        "key": "identity",
        "value": "wasm1fp7rrd...2f0yyq"
    },
    {
        "key": "owner",
        "value": "wasm1rhhjhf...y2kn7y"
    },
    {
        "key": "previousChange",
        "value": "2506"
    }
]
```

Listing 6: DIDOwnerChanged Event

それぞれの event には，previousChange というプロパティが設定されており，前回の変更が行われたときのブロック高を記録している．

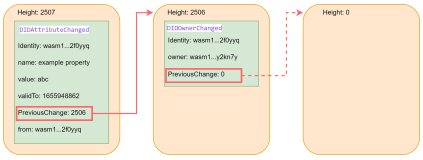

図 10: Event Recursion

そのため，図 10 のように再帰的に遡ることによって，その Identity に関わるすべての Owner の変更および attribute の追加変更を参照することができる．

しかしながら，ある Identity に関する変更履歴をブロックチェーンから直接すべて参照したい場合，ブロックチェーンに対する Query 数は変更履歴の event 数に比例して増えていくため，実用的なアプリケーションが変更履歴をブロックチェーンから参照するのは望ましくない．

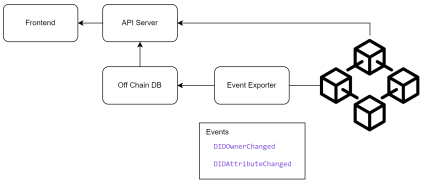

図 11: データフロー

そこで，図 11 のようにブロックチェーンのデータをオフチェーンデータベースに同期するための Event Exporter を作成し，ある Identity に関する変更履歴を取得したい場合は，API サーバーを経由してオフチェーンデータベースを参照するという方式が現実的である．オフチェーンデータベースには，MySQL 等の RDB を使うことができるため，SQL の Query で高速かつ一度の通信で欲しいデータを全部取得することができる．

5.4 DID の応用可能性

分散性や永続性，暗号学的に個人が操作していることを証明するといった特徴をもつアプリケーションは，DID と相性が良いと考えられる．DID を利用したアプリケーションの実装では，ブロックチェーン上に書き込むべきデータか，オフチェーンで管理すべきデータかを設計段階で明確に区別することが大切である．

6 展望

DID 自体は，まだ仕様策定中であり，今後の利用・実装が期待されている段階である．DID でブロックチェーンを使うに当たり，だれがそのブロックチェーンを維持

管理するのかという問題が発生する．このインセンティブ設計をうまく考えることが, 実際に運用していくにあたってもっとも重要な点であると考えられる．また, アプリケーションが複雑化すれば, ブロックチェーンに書き込む内容の一部を秘匿化したいといったニーズが生まれることが考えられる．このような秘匿化技術はまだ未熟で DID を使ってアプリケーションを構築する際に考慮しないといけない点になる．さらに, お年寄りや未成年といった自由意志を持たないエンティティの扱い方等にも課題がある．

実装面では, スマートコントラクトとやり取りをするためのクライアントライブラリの作成や DID contract への機能追加などの追加実装が課題になる．

7　結論

本研究では, まず, 現代の ID 管理の問題点を指摘し, DID が誕生した背景とともに, DID によって個人が主権的に自分の個人情報を管理する流れを整理した．次に, W3C が定義した仕様を満たすように Cosmos-SDK 及び Tendermint とそのスマートコントラクトモジュールである CosmWasm を利用して DID の実装を行った．この実装を通して, DID をアプリケーションレベルで使用するための設計要件を明らかにした．また, データベースとしてのブロックチェーンと既存の RDB との違いを比較することで, ブロックチェーンを使った実用的なアプリケーションにおけるデータフローを明らかにした．最後に, DID と相性がよいアプリケーションの特徴と, 実装における要点をまとめた．

参考文献

[1] Satoshi Nakamoto. (2008). Bitcoin: A Peer-to-peer Electronic Cash System. https:/bitcoin.org/bitcoin

[2] JBA 2016. ブロックチェーンの定義 https://jba-web.jp/news/642

[3] Naik Nitin, Jenkins Paul (2020). Self-Sovereign Identity Specifications: Govern Your Identity Through Your Digital Wallet using Blockchain Technology. https://ieeexplore.ieee.org/document/9126742

[4] W3C. 2021. Decentralized Identifiers (DIDs) v1.0. https://www.w3.org/TR/did-core/

[5] W3C. 2022. Verifiable Credentials specification. https://www.w3.org/TR/vc-data-model/

[6] 株式会社野村総合研究所,NRI セキュアテクノロジーズ株式会社. 2021. ブロックチェーン技術等を用いたデジタルアイデンティティの活用に関する研究 報告書. https://www.fsa.go.jp/policy/bgin/ResearchPaper_NRI_ja.pdf

[7] Microsoft. 2022. What are Verifiable Credentials?. https://didproject.azurewebsites.net/docs/verifiable-credentials.html

[8] LasTrust. 2020. Verifiable Credentials とは？.https://lastrust.io/2020/05/25/whatisverifiable-credentials/

[9] K. Yasuda,M. Jones. 2022. Self-Issued OpenID Provider v2. https://openid.net/specs/openid-connect-self-issued-v2-1_0.html

A　Cosmos-SDK について

Cosmos-SDK[10]は, PoS (Proof of Stake), および PoA (Proof of Authority) ブロックチェーンを構築するためのオープンソースフレームワークである．Cosmos-SDK で構築されたブロックチェーンは, 一般にアプリケーション固有のブロックチェーンと呼ばれる．SDK 内で使用可能なデフォルトのコンセンサスエンジンは Tendermint Core であり, PoS ブロックチェーンを構築するために広く使用されている．

Cosmos-SDK は, 開発者が他のブロックチェーンとネイティブに相互運用できるカスタムブロックチェーンを最初から簡単に作成できるようにすることで, Tendermint 上に安全なブロックチェーンアプリケーションの構築を可能にする．SDK ベースのブロックチェーンは複数のモジュールから構築されており, そのほとんどはオープンソースであり, 開発者なら誰でもすぐに組み込むことができる．

今日のブロックチェーンの世界における開発パラダイムの 1 つは, イーサリアムのような仮想マシンブロックチェーンである．開発は通常, スマートコントラクトのセットとして既存のブロックチェーンの上に分散型アプリケーションを構築することを中心に展開されている．スマートコントラクトは, シングルユースアプリケーションのような一部のユースケースには非常に適しているが, 複雑な分散型プラットフォームを構築するには不十分である．より一般的には, スマートコントラクトは, 柔軟性, 主権, およびパフォーマンスの点で制限されると言える．

Cosmos-SDK が提供するアプリケーション固有のブロックチェーンは, 仮想マシンのブロックチェーンとは根本的に異なる開発パラダイムを提供する．アプリケーション固有のブロックチェーンは, 単一のアプリケーションを操作するようにカスタマイズされたブロックチェーンであり, 開発者は, アプリケーションを最適に実行するために必要な設計上の決定を自由に行うことができる．また, より良い主権, セキュリティ, パフォーマンスを提供することも可能である．

B　Tendermint について

Tendermint[11]は, 複数のマシンでアプリケーションを安全かつ一貫して複製するためのソフトウェアである．安全であるということは, 最大 1/3 のマシンが任意の方法で機能しなくても合意形成が機能することを意味し, 一貫してすべての故障していないマシンが同じトランザクションログを見て同じ状態に同期する．安全で一貫したレプリケーションは, 分散システムにおいて重要な課題であり, 通貨から選挙, インフラストラクチャオーケストレーション, その他の幅広いアプリケーションのフォールトトレランスにおいて重要な役割を果たす．

Tendermint は, ブロックチェーンコンセンサスエンジンと汎用アプリケーションインタフェースという 2 つの主要な技術コンポーネントで構成されている．Tendermint Core と呼ばれるコンセンサスエンジンは, 同じトランザクションがすべてのマシンに同じ順序で記録される．ABCI と呼ばれるアプリケーションインタフェースにより, トランザクションを任意のプログラミング言語で処理することができる．開発者はあらゆるプログラミング言語で作成されたアプリケーションの BFT ステートマシンレプリケーションに Tendermint を使用することができる．

C　CosmWasm について

CosmWasm[12]は, アプリケーションレイヤーにおけるモジュールの一つで, マルチチェーン用のスマートコントラクトソリューションである．開発者は, 異なるチェーン間で, 同じコントラクトを使うことができる．すべての code は, チェーンの実態を知ることなく動作するように設計されているため, 一度 CosmWasm のコントラクトを書けば, 異なるチェーンに対して deploy することができる．

CosmWasm のもつ Actor model は信頼できる分散システムを構築する際に使われる設計手法（デザインパターン）である．基礎的な構造としては, 各 Actor (スマートコントラクト) は自分の内部状態（internal state）に関しては排他的なアクセス権を持ち, 各 Actor は直接他の Actor の内部状態に変更を加えることはできない．代わりに, Dispatcher を通してメッセージを送信（dispatch）する．

Ethereum のスマートコントラクトでは, コードをネットワークに deploy して実行するが, CosmWasm は, 最適化された wasm code の upload, 初期化, 実行の 3 ステップがある．また, CosmWasm では Ethereum の仕組みと異なり, コントラクトにトークンを送ることがコントラクトのコードを実行するトリガーになることはない．CosmWasm では, 事前に明確にコントラクトに対して実行したいことをメッセージで送る必要があるため, を設計段階で 想定される攻撃手法（Reentrancy Attacks）を軽減することができる．

[10] https://github.com/cosmos/cosmos-sdk
[11] https://github.com/tendermint/tendermint
[12] https://github.com/CosmWasm/cosmwasm